**Serious Games on the Move 08**

June 23 - 24, 2008; Cambridge, UK

# Learning programming with an RTS based Serious Game

**Mathieu Muratet**

**Patrice Torguet**

**Jean-Pierre Jessel**

IRIT, Paul Sabatier University, 118 route de Narbonne, 31062 Toulouse, France

Phone: 05 61 55 63 13

muratet@irit.fr, torguet@irit.fr, jessel@irit.fr

## ABSTRACT

This paper presents a prototype of a Serious Game that aims to entice gamers to learn computer programming by using a multiplayer real time strategy game (RTS). In this type of game, a player gives orders to his/her units to carry out operations (i.e. moving, building, and so forth). Typically, these instructions are given by clicking on a map with the mouse. The goal of this project is to encourage players to give these orders through programming. This game is intended for computer science students in higher education and can be used across both university and professional curricula. The programming languages used are C or C++.

### Keywords

Serious games, programming, real time strategy

## INTRODUCTION

In recent years, students have become less interested in science (including computer science). According to the figures given by Crenshaw et al (2008) and Kelleher (2006), the number of students registered for computer science courses has decreased. Our university has experienced the same phenomenon with a decrease of 16.6% in students studying computer science over the last four years. To recruit and retain students in our university, we have studied the potential of serious games. We believe this kind of software can increase students' interest in computer science. Therefore, we chose to use a serious game to teach programming.

## SERIOUS GAMES

The term 'serious game' is widely used, yet has many definitions. Depending on which definition is used, it is possible to consider e-learning, edutainment, game-based learning or digital game-based learning as serious games (Susi et al, 2007). The critical point of serious games is the relationship between the game and educational content, Mickael Zyda (2005) wrote that "Pedagogy must, however, be subordinate to story – the entertainment component comes first". If the game is attractive, fun, and stimulating, and encourages the user to progress, then the player automatically learns skills and absorbs a lot of information. A serious game is a video-entertainment purchase opportunity, not only for children but also for the general public. It is this approach that has enabled *America's Army* (Zyda et al, 2003) to become the first really successful serious game.

Presently, serious games exist in several fields such as education, government, health, defence,

industry, civil security and science (Social Impact Games 2006). Different types of serious games will evolve depending on the target audience:

- For the general public, serious games can be used for increasing awareness of the main problems of health, safety or environment
- For universities or companies, serious games must be able to provide a more complete and accurate knowledge depending on the user level. The goal is to enable learners to address problems in their environment in order to solve them
- For more specific training, serious game-based immersion can provide physically realistic simulations. These games use underlying mathematical models, in order to prepare the people to respond to critical situations.

Serious games must be created according to the needs and expectations of different working sectors and the public, and within the available resources (physical and financial) for their implementation. Blackman (2005) gives a synopsis of the gaming industry and its applications to the general public. Increasingly sophisticated video game graphics engines, for example, may be used for non-game applications because they often offer real-time rendering and realistic physical models. Applications such as training, and interactive visualizations and simulations already use video game technologies. We think that serious games will play an increasingly important role in education in the foreseeable future.

Thus developments in serious game applications are booming, but few of them are designed for computer science and only a minority of those are intended to teach computer programming.

## RELATED WORK

Computer science is a broad subject with many fields. Learning how to program a computer is a very important element of this subject and various software applications have been created to help students learn how to write programs.

Some of this software uses block-based graphical languages. This programming metaphor allows the student to detach him or herself from syntax and to experiment with programming. For example, *StarLogo: The Next Generation* (Klopfer et al. 2005), *Scratch* (Maloney et al 2004), *Alice2* (Kelleher et al. 2002) and *Cleogo* (Cockburn and Bryant 1998) use this approach.

Another method involves using competition to motivate students, which is the basis of both the *Robocode* project (Nelson 2001), and the international event *RoboCup* (2007), and which involve programming Artificial Intelligences (AI) for robots.

Other attempts to hook the player and encourage him or her to learn programming use video games. The *WISE* (Wireless Intelligent Simulation Environment) project (Cook et al. 2004), for example, is an interactive game environment which combines virtual and physical games. *Colobot* (Epsitec, 2007) is the only example, as far as we know, of a complete video game which mixes interactivity, story, and programming. In this game, the user must colonize a planet using robots that s/he is able to program.

As noted above, we think that serious games are an effective teaching method. With the exception of *Robocode*, *WISE* and *Colobot*, we do not consider the software packages listed above as video games. Moreover, *Robocode*, *WISE* and *Colobot* do not fill our criteria: *Robocode* lacks interactivity, as the player is inactive during the simulation and is merely a spectator of his/her AI; *WISE* requires a number of resources (i.e. space, robots, and so on); *Colobot* lacks a multiplayer mode, which is useful to motivate players by offering collaborative and competitive options (Johnson and Johnson 1994).

Nevertheless, our work relies on these tools. We propose to teach programming via an entertainment, interactive and multiuser platform. For these reasons, we chose a popular gaming genre and an existing multiplayer video game. We upgraded it to enable control of entities through programming. In other words, our tool is a multiplayer video game where programming is a feature of the game.

## EDUCATIONAL CONTENT

Our goal is to use a serious game to provide alternative and/or complementary methods to traditional systems for teaching students how to program. The game is intended for computer science students in higher education. It allows them to implement several styles of coding such as imperative, object-oriented, event or parallel programming using C or C++ programming languages (commonly used in industry nowadays). These characteristics make it possible to use this technique in all types of teaching.

From the user's perspective, the compiled code is dynamically and interactively integrated into the game. The game consistency is kept with a set of linkage and synchronization mechanisms which are totally invisible to the player. This leaves the player free to concentrate on the game.

The game supports different degrees of abstraction which allows it to be used at any level of training. Beginners, for example, only have access to a minimal interface which is used to give simple orders to units. On the other hand, more experienced students can have access to the full engine implementation, and they can, therefore, apply complex concepts to develop a sophisticated application.

## SYSTEM DESCRIPTION

To support our system, we chose to use a type of game familiar to players: Real Time Strategy (RTS). In this game category, the player leads an army composed of units. S/he can interact with the virtual environment by giving orders to his/her units to carry out operations (i.e. move, build, and so on). Usually, these orders are given by clicking on a map with the mouse. Our goal is to encourage the player to give these orders through programming.

Since we did not intend to develop a new RTS engine, we looked for an existing game to use as a starting point. Any such game must both meet our expectations and be able to incorporate our improvements. Fortunately, some open source projects exist, such as *Open Real-Time Strategy* (ORTS) and the *Spring* project (Spring 2006), both of which are multiplayer 3D real-time strategy games (Figure 1).

*ORTS* (Buro 2002; Buro and Furtak 2005) has been developed to provide a programming environment for discussing problems related to AI. This game is designed to allow the user to easily program and integrate his/her AIs, *ORTS* also incorporates a multiplayer mode, which is important to us because, in the future, we intend to focus our serious game on competition and collaboration, as we think that this will increase students' motivation and investment. Eventually, we would like to develop a massively multiplayer online game (MMOG) version, and thereby create a persistent environment where it would be continually possible to do programming in a playful way.


**Figure 1. On the left: ORTS. On the right: Spring**

Recently, we have also managed to integrate our module into another game engine, *Spring*, in order to ensure its independence from *ORTS*. This engine is completely different from *ORTS* both in its network architecture and in its internal design. However, we were able to transplant our module onto this game with only minor modifications.

## TECHNICAL ASPECTS

Our objective is to allow players to write code and to integrate it into the game, where it will be run. In this way the player can observe the implementation of his/her code through the behaviour of his/her units in the video game. But in *ORTS*, each code modification results in the interruption of the execution of the program, which is then rebuilt and restarted for changes to take effect. This is a key element of compiled programming languages such as C++, which is used by *ORTS*. In the first stage of our project, therefore, the first issue to resolve was how to integrate player's code into the game engine without having to stop, or recompile it. This improvement allows greater interactivity because the player is able to modify, compile and integrate his/her code without stopping the game which consequently maintains its progress and coherence.

Moreover, we also wanted programming beginners to be able to use the prototype. Their main concern is to understand the concepts and principles of this discipline. So, to assist them, we have hidden any additional difficulties related to the complexity of the game engine so that they can focus on their goals.

To solve these problems, we could have chosen a scripting language but for performance, we chose to use a dynamic library and to design a simple application programming interface (API) to the game engine.

### Dynamic Library

The semantics of the term ‘dynamic library’ defines its purpose. The library provides functions that can be called up and executed by the program that uses it. As for the concept of dynamic, it determines that the library can be loaded, used and discarded during execution.

In our application, the library contains the code written by the player and defines the behaviour of his/her units. The game uses this library to identify which actions to carry out. Each modification of the library triggers the loading of the new AI. If no library is accessible, no automatic processing will be executed. Through this principle, the code containing the behaviour of units is completely independent of the game. In this way, the player can change his/her code and recompile it as a library to be automatically integrated into the game. Although the library is normally sufficient in itself, our application requires access to the *ORTS* toolbox to manipulate library data. It was therefore necessary to modify the game engine to enable the library to access elements of the game.

The library implements the player’s code in a thread in order to allow the client to remain active and responsive to the user’s (or server’s) actions. Thus, complex behaviours can be programmed in the library without influencing the game’s performance. As parallel programming introduces additional difficulties in the execution, this programming type requires the implementation of synchronization mechanisms between various processes in order to ensure the consistency of shared data. The player, however, is not aware of this.

We have also tried to ensure system reliability by protecting it against player’s coded bugs. Since the player is involved in a learning process, it is highly likely that he/she will make mistakes. These errors can cause system failures (i.e. segmentation faults, for example), or raise exceptions, however, as noted above, we have specified that the library is performed in a thread. This feature has been used together with system signals to identify errors triggered in the library. So, when an interruption occurs in the AI, only the thread running the library is interrupted. Information is then given to the user to warn him/her of the error type that stopped his/her AI. In this way, game execution is not dependent on AI malfunctions.

The use of a dynamic library has an added advantage in that it hides from the player the complexity of the video game. As noted above, the player must write the code corresponding to the behaviour of his/her units. As a general rule, to be able to change a part of a program requires analysis of the structure, organization and operation of the application. The dynamic library helps the player by extracting AI from the game. In this way, the user is not aware of the difficulties associated with the integration of his/her code in the game engine.

### Development environment

The dynamic library gives the player the opportunity to amend his/her code in an interactive way. It assists his/her work by hiding the game's complexity. However, use of the application remains tedious, because of the game's architecture and its file tree. To help the player further, it is necessary to make the software more intuitive. For this reason, we have developed an interface called the Development Centre (DC) (Figure 2).

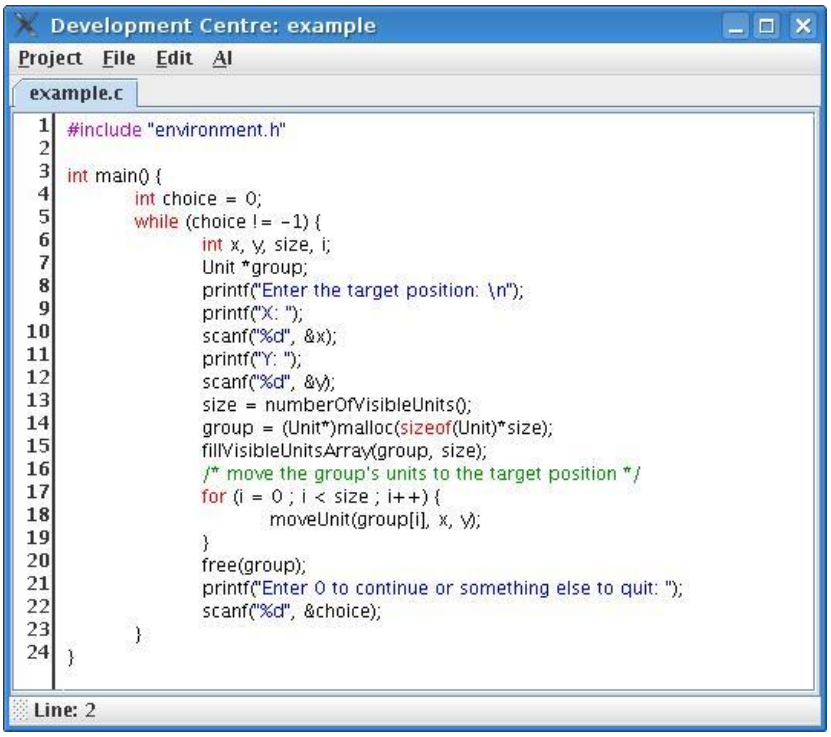


**Figure 2. The Development Centre**

The development centre is a component that facilitates the design and manipulation of the player's objectives. It is automatically launched at the beginning of the game to manage the projects through a series of menus. However, the DC is independent and is not necessary for the operation of *ORTS* and vice versa. Its purpose is to hide the mechanisms of synchronization and linkage between the player's code and the game. Thus, the player starts by creating his/her classic function "int main (){...}" as if his/her code was independent of the game. S/he can then use one set of functions (defined according to the programming knowledge of the player - the goals, for instance) to manipulate entities in the game. The player can then easily compile and inject his/her code into the game engine and see the results.

### Overview

Our application is composed of three entities (Figure 3): the game engine, the dynamic library, and the DC. In order to develop our software, we had to modify the game engine by adding a façade class to allow access to the game data (called "State"), a synchronization tool ("Monitor"), and a class that manages the dynamic library ("Loader"). These entities are described in detail below. The dynamic library consists of an interface that allows its use (i.e. the User Code Management Interface (UCMI) which permits control of the behaviour of the thread), and a thread that executes the player's code.

The "Loader" is designed to manage the dynamic library. If the dynamic library is created or is changed it reloads it. The "Loader" is also used to pilot the thread through the UCMI to maintain the same version in both the library and the thread.

The "Monitor" synchronizes the thread and the "Loader" in a way that respects the integrity and consistency of the game progress. The player's code, carried out in the thread, can access the game information through the "State" class. This class serves as an entry point into the game data.

Finally, the DC allows the player to modify his/her code, to compile it in a dynamic library, and to notify the game engine that the code is changed and it is time to update it.

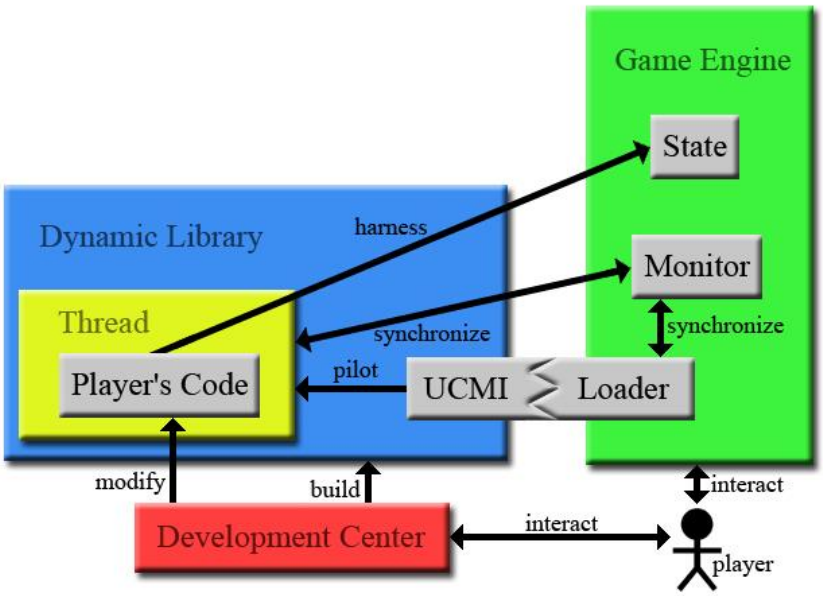


**Figure 3. Architecture**

## CONCLUSIONS AND FUTURE WORK

In this paper we have described a serious game prototype designed to teach computer programming in a fun and interactive way. This approach is motivated by the decrease in the numbers of students studying computer science. The application consists in adapting a game engine so that it can integrate some code during its execution. This software allows, unlike other applications for learning how to program, the interactive manipulation of two widely used programming languages: C and C++. This prototype enables novice or experienced students to enjoy themselves while developing computer programs. Furthermore, we have established that our adaptation could be easily integrated in other RTS game engines (such as *Spring*). This allows the selection of the best game according to the learning objectives, and the development of the game to keep pace with the rapid evolution of video game standards.

However, the serious game, in itself, does not yet exist. We still need to create a scenario which will involve the player so that s/he is encouraged to learn programming.

The next step is to conduct experimental field work. We have started to work with collaborators from our university's learning and teaching support service; a didactic specialist and professors and students. We want to discover the breadth of teaching applications supported by our system as well as the range of potential audiences and teaching methodologies. Analysis of our experiments will explore and resolve potential issues concerning usability and effectiveness. At the same time, it will be important to determine how users switch between game play and coding elements.

We plan to develop a scenario builder, which will enable teachers to easily create fun lessons, including educational aspects, in order to engage and motivate the students.

RTS mainly work on Peer–to-Peer (P2P) architectures and the complete simulation is duplicated for each peer. For every simulation step, each peer synchronises its simulation with other peers. This architecture is not scalable and limits the number of players. *ORTS,* however, has a client-server architecture where the server runs the simulation. Clients only see a small part of the world, which could be interesting in order to transform ORTS into a MMOG system. Such an MMOG would allow thousands of users to share their experiences in a persistent virtual world for collective training. However, the subject of MMORTS has been little explored, offering opportunities for exciting research.

Finally, the DC allows easy use of the software, but it could be improved in order to facilitate interaction with the virtual environment. It would also be interesting to set up an optional system of block-based graphical coding with drag and drop as in *Alice2* or *StarLogo: The Next Generation*. This would help beginners, relieving them of the C or C++ syntax.

All these enhancements will allow our application to become a massively multiplayer online serious game (MMOSG) and, more precisely, a massively multiplayer online serious real time strategy game (MMOSRTS) where the only boundaries are the imaginations of the players' and teachers'.

## REFERENCES


Blackman S, (2005), *Serious games...and less!*, in SIGGRAPH Computer Graphics, 39(1), p.12-16

Buro M, (2002), *ORTS: A Hack-Free RTS Game Environment*, in 3rd International Conference on Computers and Games. Edmonton, Canada, 25-27 July 2002, Springer: Berlin

Buro M & Furtak T, (2005), *On the development of a free RTS game engine*, in 1st Annual North American Game-On Conference, Montreal, Canada, 22-23 August 2005

Cockburn A & Bryant A, (1998), *Cloego: Collaborative and Multi-Metaphor Programming for Kids*, in 3rd Asian Pacific Computer and Human Interaction, Shonan Village Center, Japan, 15-17 July 1998, IEEE Computer Society: Washington DC USA

Cook DJ, Huber M, Yerraballi R & Holder LB, (2004), *Enhancing Computer Science Education with a Wireless Intelligent Simulation Environment*, in Journal of Computer in Higher Education, 16(1), p. 106-127

Crenshaw TL, Chambers EW, Metcalf H & Thaklar U, (2008), *A case study of retention practices at the University of Illinois at Urbana-Champaign*, in 39th ACM Technical Symposium on Computer Science Education. Portland, Oregon USA, 12-15 March 2008

Epsitec, (2007), Colobot, available at: http://www.ceebot.com/colobot/index-e.php [accessed 21 September 2007]

Johnson RT & Johnson DW, (1994), *An overview of cooperative learning*, originally published in Thousand J, Villa A & Nevin A (Eds.), Creativity and collaborative learning, Baltimore: Brookes Press

Kelleher C, (2006), *Alice and The Sims: the story from the Alice side of the fence*, in The Annual Serious Games Summit DC Washington, DC, USA, 30-31 October 2006

Kelleher C, Cosgrove D, Culyba D, Forlines C, Pratt J & Pausch R, (2002), *Alice2: Programming without Syntax Errors*, in 15th Annual Symposium on the User Interface Software & Technology, Paris, France, 27-30 October 2002

Klopfer E & Yoon S, (2005), *Developing Games and Simulations for Today and Tomorrow's Tech Savvy Youth*, in TechTrends: Linking Research & Practice to Improve Learning, 49(3), p.33-41

Maloney J, Burd L, Kafai Y, Rusk N, Silverman B & Resnick M, (2004), *Scratch: A Sneak Preview*, in 2nd International Conference on Creating Connecting, and Collaborating through Computing, Keihanna-Plaza, Kyoto, Japan, 29-30 January 2004, IEEE Computer Society: Washington DC USA

Nelson M, (2001), Robocode, available at: http://robocode.sourceforge.net/ [accessed 17 April 2007].

RoboCup, (2007), RoboCup, available at: http://www.robocup.org/ [accessed 09 April 2007]

Social Impact Games, (2006), Entertaining Games with Non-Entertainment Goals, available at: http://www.socialimpactgames.com/ [accessed 02 February 2006]

Susi T, Johannesson M & Backlund P, (2007), *Serious Games – An Overview*, in Technical Report HS-IKI-TR-07-001, School of Humanities and Informatics University of Skövde, Sweden, 5 February 2007

Spring, (2006), The Spring Project, available at: http://spring.clan-sy.com/ [accessed 2 February 2007]

Zyda M, (2005), *From Visual Simulation to Virtual Reality to Games*, in Computer, 38(9), p.25-32

Zyda M, Hiles J, Mayberry A, Wardynski C, Capps M, Osborn B, Shilling R, Robaszewski M & Davis M, (2003), *Entertainment R&D for Defense*, in IEEE Computer Graphics and Applications, 23(1), p.28-36, 2003